Jackson State University
Department of Computer Science
CSC 438/539 Systems and Software Security, Spring 2014
Instructor: Dr. Natarajan Meghanathan
Project 2: Simulating the TOCTTOU Vulnerability in a Linux-C++ Environment

Due Date: March 26, 2014, 7.30 PM Max. Points: 100

Project Objective: The objective of this project is to demonstrate the TOCTTOU (Time of Check to Time of Use) vulnerability in Linux environment using a simple file updater program written in C++.

Introduction to TOCTTOU Vulnerability

TOCTTOU stands for Time of Control to Time of Use. This type of vulnerability occurs when a user is granted permission to access to a resource or file at a particular time and if access to the file or resource is taken away from the user, the user may still access the file or resource if the user has never relinquished control over the asset.

In this activity, you will simulate this kind of vulnerability using a text file in an Ubuntu virtual machine. You will create two users in the operating system: User A and User B (see the details below for the exact naming convention for the user names). User A will create a text file and grant permission to everyone to read and write to it. User B will then access this file via a simple program (written in C++). While User B is accessing the file, User A will revoke User B's permission to read from and write to the file. Even though User B loses permission to access the file, User B has *already* been granted permission to access the file, and is able to continue reading and writing to the file until it is released.

It is important that you read this entire document first, as some of the steps are time-sensitive. You will need to know in advance what you will need to do in order for the activity to work properly.

What to Submit (Submit the recorded video through GoogleDrive using your JSU email address; after uploading the video, e-mail the link to natarajan.meghanathan@jsums.edu):

After you setup the VM, install g++, type and compile your program in user B's account, start the recording and record as follows:

(1) Login to User A's account; create a text file (owned by user A) and set its permissions such that others can read and write to it.

(2) Login to User B's account in another terminal. Launch the file updater program to write to the text file of user A.

(3) As it is running, go to the terminal where you have logged in as User A and displays the contents of the text file. The text file's contents should reflect the updates from user B's file updater program. Now, let user A to change the permission for others to be not able to write to the file (the read permission is still granted).

(4) Return to the terminal for User B and show that the program is still running without any problem/error reported. After a couple of minutes, go the terminal for user A and display the contents of the text file. Explain the TOCTTOU vulnerability that is behind the fact that the contents of the file continuing to be updated.

(5) After the file updater program at user B stops, launch it again. Observe what happens when you try to start fresh and run the program at user B. Explain the reason behind what you observe.

Desktop Video Recording

You could try using one of the **desktop recording software** (or anything of your choice):

CamStudio: http://sourceforge.net/projects/camstudio/files/legacy/

Debut: http://www.nchsoftware.com/capture/index.html

Creating the Programming Environment: Ubuntu/Linux VM

Task 1:

You will use the Ubuntu VM that you installed in Oracle Virtualbox or VM Player for Project 1. If you have not already installed Ubuntu VM in Oracle Virtualbox, read the description for Project 1 to complete this task.
To install C++ on the Ubuntu VM, follow the instructions in Task 2.

Task 2: Installing C++ on your VM

Open a terminal by using Applications --> Accessories --> Terminal. Type "sudo apt-get install g++" and press enter. You will then be prompted for your password. This will install the g++ compiler.

```
security@security-desktop:~$ sudo apt-get install g++
```

Task 3: Creating the Two User Accounts

Make sure you install C++ on your VM, before you create the two user accounts and logging in as the two users.
Open a terminal by using Applications --> Accessories --> Terminal. Let the two user accounts you create be named using the following convention.. [FirstInitial][LastInitial]_UserA and [FirstInitial][LastInitial]_UserB

In my case, I am going to create the two user accounts with name NM_UserA and NM_UserB.

You would have to use the "--Force-badname" option to let the system to create the usernames of your choice.

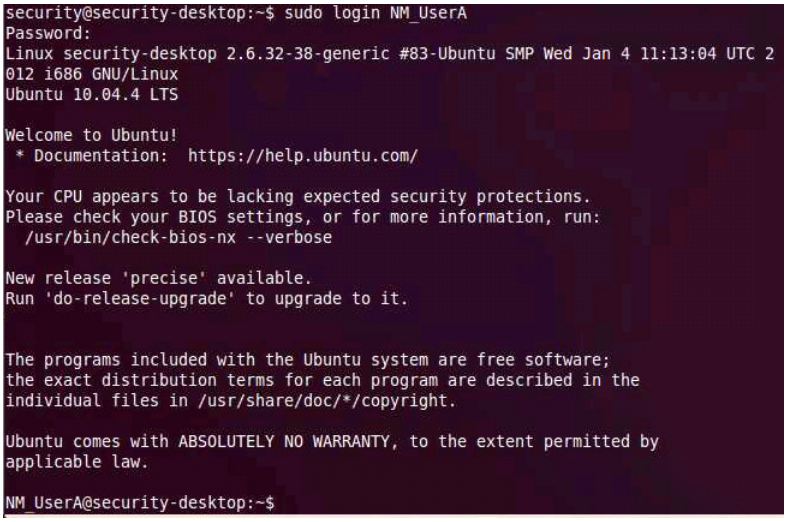
You will create a user account as shown in Figure 1.

```
security@security-desktop:~$ sudo adduser --force-badname NM_UserA
Allowing use of questionable username.
Adding user `NM_UserA' ...
Adding new group `NM_UserA' (1001) ...
Adding new user `NM_UserA' (1001) with group `NM_UserA' ...
Creating home directory `/home/NM_UserA' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for NM_UserA
Enter the new value, or press ENTER for the default
        Full Name []: NM_UserA
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
security@security-desktop:~$
```

[NM_UserA, as above] and [NM_UserB, when the procedure is repeated]

Following the above approach, you can create user accounts NM_UserA and NM_UserB. Make sure to remember the usernames and passwords you setup for the two user accounts that you create.

Task 4: User A: Creating the text file to update

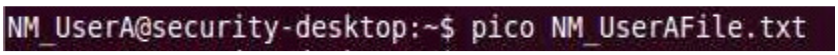


You will now login as User A using the following sudo command:

sudo login NM_UserA

Inside the NM_UserA account, create a text file by name NM_UserAFile.txt, depending on the naming convention that corresponds to you.

Use an editor of your choice to create the text file. I use the pico editor to create the text file. In the pico editor, after typing the contents, I press Ctrl+X to save and exit. The contents of the text file should be your name and the name of the course.



After you type the contents of the file, save and exit; set the permissions for **others** to be able to "read and write" to the file. Capture this as screenshot Figure 3. Note that User A considers User B to belong to the 'others' category.

Then, use the *ls -l* command to display the permissions of the files in the folder.

Task 5: User B: Creating the C++ program to update the text file

Open another terminal. You will now login as User B using the following sudo command:

sudo login NM_UserB

Again, using an editor of your choice, type the C++ program to update the text file you created in user A's account. The following description will guide you through the development of the C++ program. This C++ program will do several things. First, it will open _UserA's text file, read the contents and print them out to the screen. Then, it will open the text file for writing. It will initiate a five-minute loop in which a new line is written to the file every 30 seconds. Once the five minutes has passed, the file will be closed, and the program will end.

You need to include several C++ and C header files as well as the namespace std, as follows:

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
#include <unistd.h>

using namespace std;
```

I have commented the various statements you would need to develop a complete C++ program to read and update (append) the contents of a text file. Replace the comments with the appropriate code.

```
int main(){

  // Declare an object 'infile' of class ifstream

  // Call the open method on the infile object and pass the path of the NM_UserAFile.txt in NM_UserA's account


  /* Call the fail() method on the infile object and print an error message like "file cannot be opened for reading" in case you cannot open the file... Terminate
    (return from) the program and do not execute further. */

  // Declare an object by name 'line' of class string


  // Call the getline() method and pass the infile and line objects as arguments and print the contents of the line. Repeat this in a while loop

  // Call the close() method on the infile object

  // Declare an object 'outfile' of class ofstream

  // Call the open method on the outfile object and pass the two arguments: (1) path of the NM_UserAFile.txt in NM_UserA's account and (2) ios::app to append to the file


  /* Call the fail() method on the outfile object and print an error message like "file cannot be opened for writing" in case you cannot open the file... Terminate
    (return from) the program and do not execute further. */
```

```
 // Run the following in a for loop, for 10 times

 for (int i=1; i<=10; i++){

    // get the current system time in milliseconds since Jan 1, 1970, using the time(0) function and store it in an object of name currentTime of class time_t.

    // get the pointer reference 'now' of struct type tm to currentTime as follows
    struct tm *now = localtime(&currentTime);

     // Create a string object (say by name timeString) of the date/time in 'now' by calling the asctime( ) function and pass the 'now' struct pointer as the argument

     // Direct (<<) the contents of the string timeString to the outfile object

     Call the sleep( ) method and pass 30 as the argument - to indicate the program should sleep for 30 seconds for every iteration of the loop

  }

  // Call the close( ) method on the outfile object


 return 0;


}
```

Task 6: Run User B's File Updater C++ Program

Now you have written the complete C++ program which opens a file for reading, reads and prints out its contents, and closes the file. The program then opens the file for appending (writing to the end), writes the current time, waits for one minute, and loops ten times (writing the current time to the file once a minute for five minutes), and closes the file.

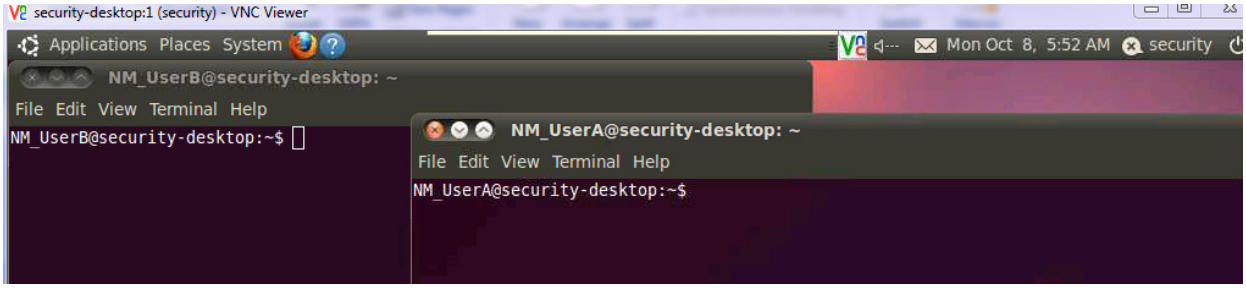
To compile the file, type g++ followed by (a blank space and) the name of your .cpp file (including the .cpp extension), and press enter. If you are presented with any errors, you will need to find and correct them.

Once the file has compiled without any errors, use the *cat* command to print the contents of the complete the file updater C++ program that you had just developed.

```
NM_UserB@security-desktop:~$ g++ fileUpdaterNM.cpp
```

Once the file is compiled, you will need to run the file, but before you do, make sure that you know how to change file permissions (review Project 2, if needed). Once you begin running the file, you will have five minutes in which to change the file's permissions in order to get the full effect of this activity.

To run the file, type g++ followed by (a blank space and) the name of your C++ class (excluding the .cpp extension), and press enter. You will see the contents of the file output to the screen, and then you will not see any activity in the terminal window until the program is complete.



Task 7: User A: Changing the Permissions for User B

While the file updater C++ program is running at the terminal for User B, go to terminal that you have opened for User A; wait for a minute or so, and then run the *cat* command to display the contents of the NM_UserAFile.txt file. What is that you observe? Explain the reason behind what you observe.

Now, use the *chmod* command to remove the write permissions for **others**.

Use the *ls -l* command to display the permissions of the files in the folder for User A.

Let the C++ program to complete running in the terminal for User B.

In the terminal for User A, use the *cat* command to display the contents of the NM_UserAFile.txt file.

Task 8: User B Running the File Updater C++ Program (after User A removed the write permissions to the text file)

Now, in the terminal for User B, run the file updater C++ program again. You should see an error message. Capture the error message as screenshot.

Explain why you now got an error message now and not in Task 6.